Efficient Low-Rank Convolutional Neural Networks

Sri Jaladi, Ishan Khare, Bar Weiner

Table of contents

01

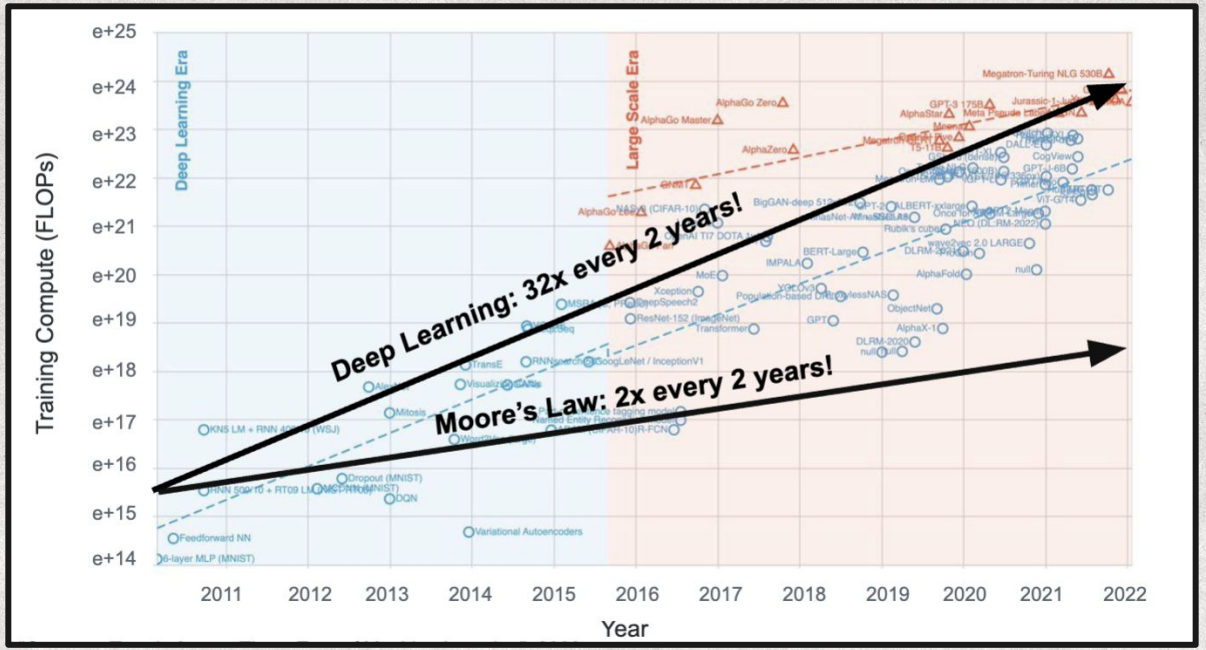
Problem statement:

How does utilizing low-rank convolutions affect a CNN's performance in computational time, memory, and accuracy?

02

Background

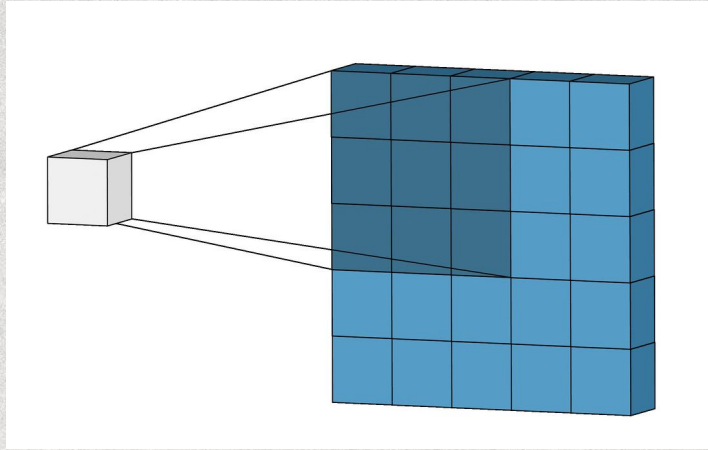
Motivation



Deep learning compute requirements are growing faster than hardware growth creating bottlenecks in computational efficiency.

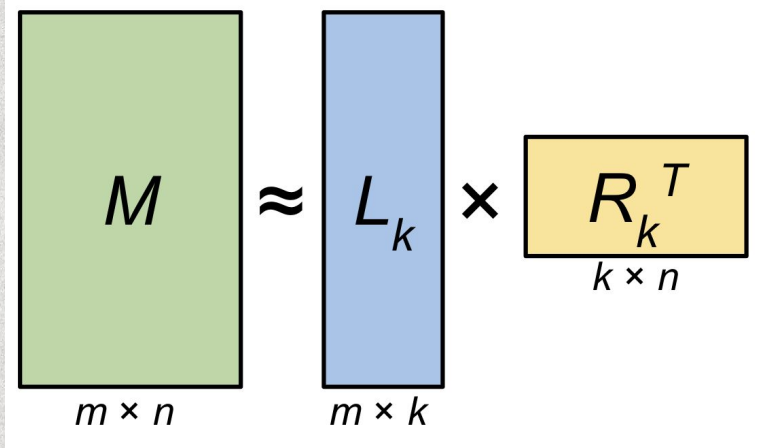
Convolutions

- Slide a size K x K trained kernel across our input image
- Process of repeatedly sliding is highly time-inefficient
- We show optimizations: instead of using a full K x K convolution, for large K, we use a low rank representation.



Existing Approaches

- **Low-Rank approximation (LoRA)** for rank-reduction of matrix M.
- LoRA (used by OpenAI + others for LLMs)
  - 1) train full model,
  - 2) approximate, and
  - 3) fine-tune.
- We propose an alternative to LoRA, where we:
  - 1) enforce rank-n matrices during training
  - 2) inference with efficient weights



03

Hypothesis:

Constraining convolution training to a rank-N space, leads to: well-trained, well-performant models with parameter and time efficiency.

04

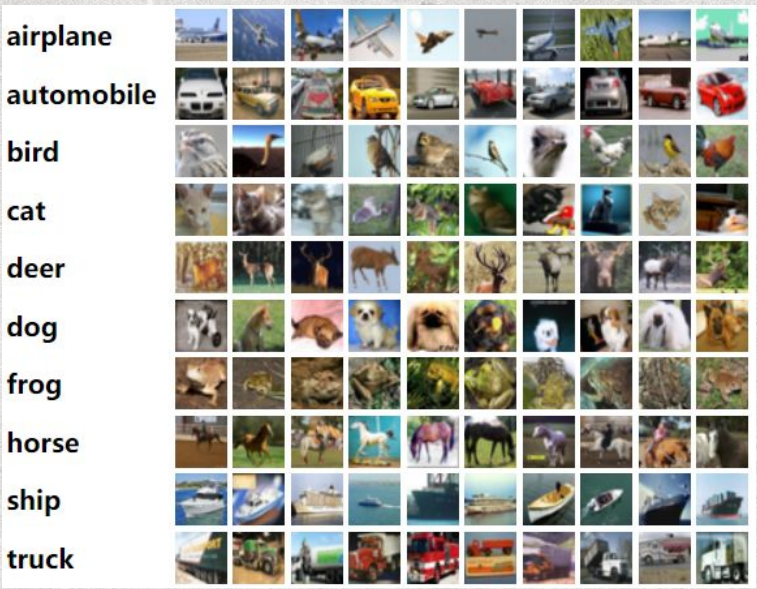
Methodology & Experiments

Experimental Setup

- Run a classification task on the CIFAR-10 Dataset -> Split into Train, Test and experiment.

For each experiment we do the following:

1. Batch learning (64)
2. Train for 10k iterations
3. Test on the entire test set
4. Log important information
5. Repeat 50 different seeds each time



CIFAR -10 Dataset

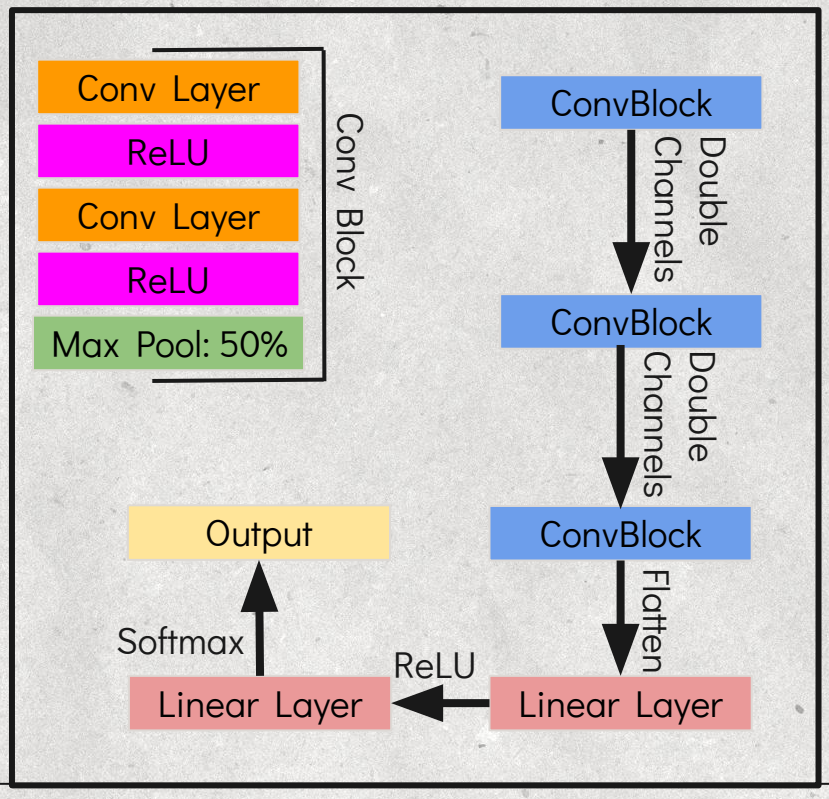
Experiments Conducted

In total we ran experiments with the following architectures:

1. Baseline (default architecture)
2. Using dropout (in between each convolutional **block**)
3. Using Batch Norm (in between each convolutional **layer**)
4. Using **Both** dropout and batch norm

For each experiment, we conducted experiments using kernel sizes of 3,5,7.

For each kernel size, we tested all rank-N training spaces up till the size



Model Architecture

Low Rank Convolutional Layer

- We create our own custom Low-Rank Convolutional Layer which forces its weights onto a Rank-N space.
- We manually deal with proper initialization, maintenance, and Toeplitz Multiplication of the Rank-N space throughout training, but utilize PyTorch's auto-gradients.
- We then build on this by developing our own Convolutional Block so that each layer is a Rank-N enforced Low-Rank Convolutional Layer

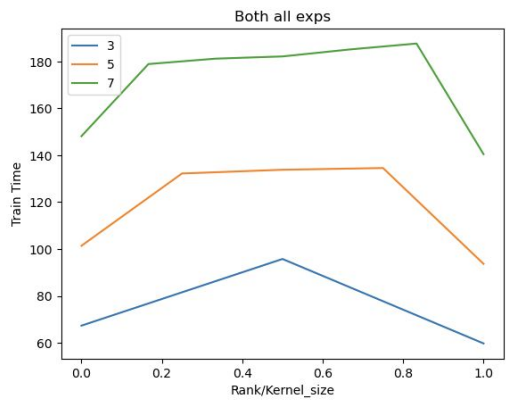
$$\begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n & 0 & 0 & 0 & \cdots & 0 \\ 0 & x_1 & x_2 & x_3 & \cdots & x_n & 0 & 0 & \cdots & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & \cdots & x_n & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & x_1 & \cdots & x_{n-2} & x_{n-1} & x_n & 0 \\ 0 & \cdots & 0 & 0 & 0 & x_1 & \cdots & x_{n-2} & x_{n-1} & x_n \end{bmatrix}.$$

Toeplitz Matrix - allows for efficient, parallelizable matrix multiplications

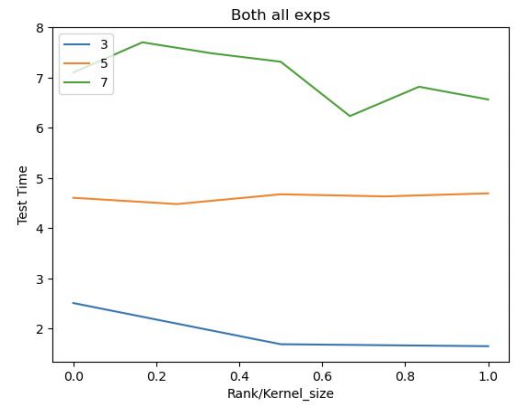
05

Results & Analysis

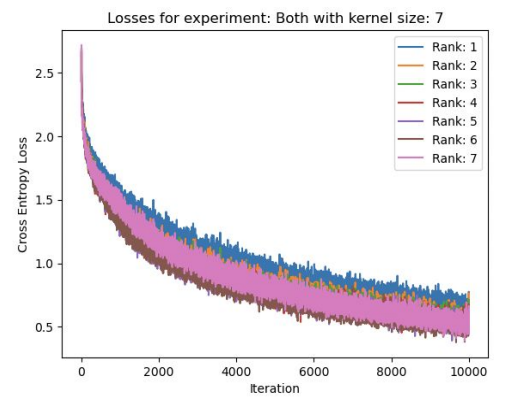
Timing Results



Increasing kernel size, increases training time. However, baseline convolutions (x = 1.0) outperform Rank-1.



Kernel Size has minimal to no impact on testing time. Toeplitz method is promising.

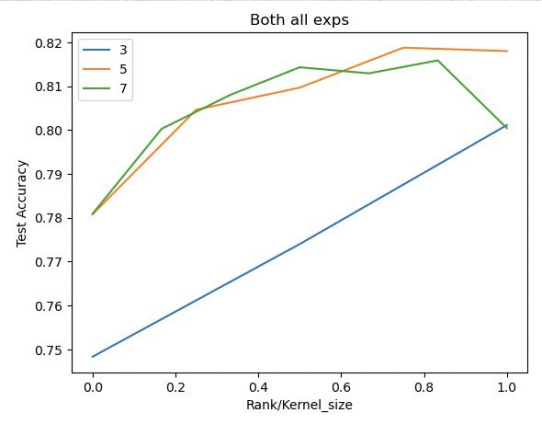


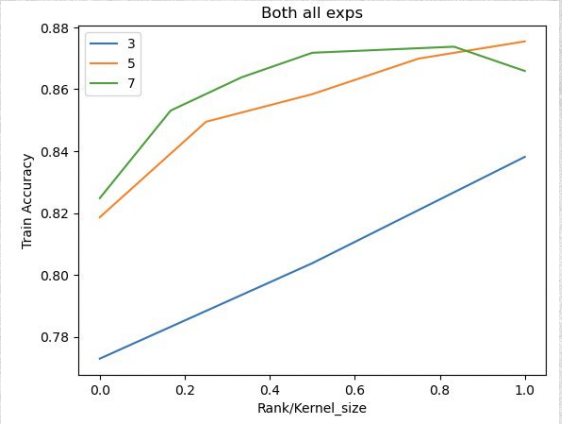
Some Ranks reach better performance more quickly than the baseline (Rank 7)

Time and Space

• Were able to effectively cut the space of CNN weights down from O(N^2) to O(N) • However, time often varied due to other model components	• Time for testing was rather random in which performed better due to convolutions not being the bottleneck factor.
• Due to linear layers and activations, the convolution wasn't necessarily the bottleneck. • General positive correlation between rank and time for training	• Our training implementation is comparable to PyTorch and thus promising for future work.
• Our rank-1 training (naively implemented) rivaled that of PyTorch's optimized CNN	• Component of interest: sometimes smaller ranks achieve better performance more quickly; analyzed further later

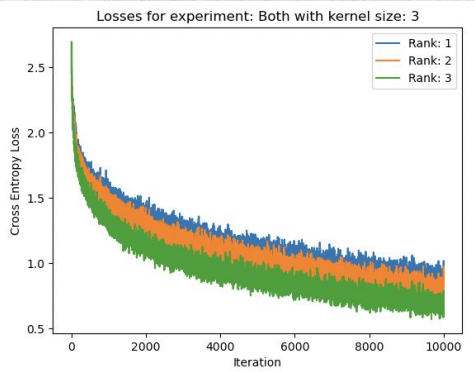
Train and Test Accuracy vs Rank/KS

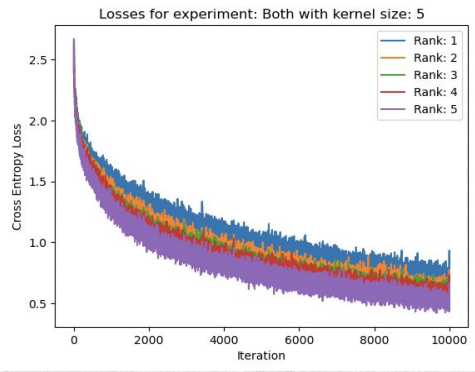


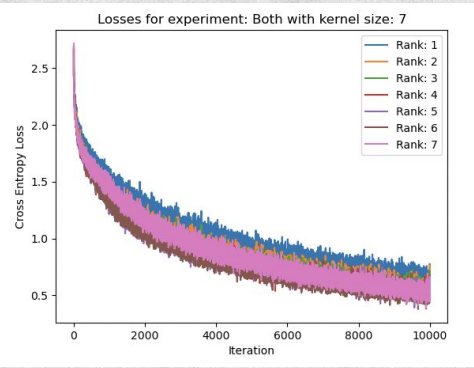


- Increasing Kernel Size generally increases accuracy.
- Rank-1 is only 2% off PyTorch default convolutions: a promising tradeoff.
- Generalizability is slightly better: the difference between train and test performance is better when using Low-Rank Convolutions
- Rank–6 outperforms PyTorch default convolution at Kernel Size 7

Loss vs Kernel Size







- As kernel size increases, lower rank matrices obtain performance quicker in comparison to PyTorch default Convolution. *E.g.* in Kernel Size 7, Rank 6 trains more efficiently than baseline.

Performance Learnings

Findings:

- Performance gaps even to Rank-1 are fairly small (only 2% in large kernels).
- Small ranks can have advantages during training, especially when dealing with larger kernels and thus larger models.
- The smaller rank with a larger kernel size seems to be effective due to its ability to capture a large receptive field while minimizing parameters, making it generalizable.
- Performance increase over the iterations.
- As the size of the kernel increases, the lower ranks perform better earlier and earlier in training compared to the baseline.
- With a kernel size of 3, the gap early in training is quite large, but with a kernel size of 7, ranks 5 and 6 outperform the baseline.

06

Discussion & Conclusion

Final Thoughts & Discussion

- **Primary goal**: identify the tradeoff between the computational efficiency, time, and performance of the model when using our newly proposed low-rank convolutions.
- Overall success as we proposed a new form of training, keeping the entire set of weights entirely in the rank-N space.
- Maintained accuracies, even with very low rank spaces, indicating that convolutions can be made more efficient in parameter count.

Explicitly, we effectively cut our convolutional trainable parameters from O(N^2) to O(N) and in the process saw positives in the realm of:

1. Testing time (approx. equal to default Pytorch)
2. Speed to reach accuracy
3. Testing accuracy
4. Model generalizability
5. Relative training time

Future Work

1. Optimizing the training and testing process to **surpass PyTorch results**.
2. Test this approach with **larger CNNs** or other models.